Introduction to scikit-Learn(sklearn)

This notebook demonstrates some of the most useful functions of the beautiful scikit library

What we are going to cover

0. An end-to-end scikit-learn workflow
1. Getting the data ready
2. Choose the right estimator/algorithm for our problems
3. Fit the models/algorithm and use it to make predictions on our data
4. Evaluating a model
5. Improve a model
6. Save and load a trained model
7. Putting it all together

In [12]:
```
import numpy as np
```

0. An end-to-end scikit-learn workflow

In [13]:
```
# 1. Getting the data ready
import pandas as pd
heart_disease = pd.read_csv('heart-disease.csv')
heart_disease
```

Out[13]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [14]:
```
# create x (features matrix)
x = heart_disease.drop('target', axis = 1)

# create y (labels)
y = heart_disease['target']

```

In [15]:
```
# 2. choose the right model and hyperparameters
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()

# we'll keep the default hyperparameters
clf.get_params()
```

Out[15]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [16]:
```
# Fit the model to the training data
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

In [17]:
```
clf.fit(x_train, y_train)
```

Out[17]:
```
RandomForestClassifier()
```

In [18]:

```
x_train
```

Out[18]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
106	69	1	3	160	234	1	0	131	0	0.1	1	1	2
178	43	1	0	120	177	0	0	120	1	2.5	1	0	3
269	56	1	0	130	283	1	0	103	1	1.6	0	0	3
153	66	0	2	146	278	0	0	152	0	0.0	1	1	2
186	60	1	0	130	253	0	1	144	1	1.4	2	1	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...
129	74	0	1	120	269	0	0	121	1	0.2	2	1	2
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2
116	41	1	2	130	214	0	0	168	0	2.0	1	0	2
140	51	0	2	120	295	0	0	157	0	0.6	2	0	2
146	44	0	2	118	242	0	1	149	0	0.3	1	1	2

242 rows × 13 columns

In [19]:

```
# make a prediction
y_label = clf.predict(np.array([0, 2, 3, 4]))
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid featur
e names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6928\3217224712.py in <module>
      1 # make a prediction
----> 2 y_label = clf.predict(np.array([0, 2, 3, 4]))

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    806             The predicted classes.
    807         """
--> 808         proba = self.predict_proba(X)
    809 
    810         if self.n_outputs_ == 1:

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict_proba(self, X)
    848         check_is_fitted(self)
    849         # Check data
--> 850         X = self._validate_X_predict(X)
    851 
    852         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    581             raise ValueError("No support for np.int64 index based sparse matrices")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately, **check_params)
    564                 raise ValueError("Validation should be done on X, y or both.")
    565         elif not no_val_X and no_val_y:
--> 566             X = check_array(X, **check_params)
    567             out = X
    568         elif no_val_X and not no_val_y:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    767             # If input is 1D raise error
    768             if array.ndim == 1:
--> 769                 raise ValueError(
    770                     "Expected 2D array, got 1D array instead:\narray={}.\n"
    771                     "Reshape your data either using array.reshape(-1, 1) if "
```

```
ValueError: Expected 2D array, got 1D array instead:
array=[0. 2. 3. 4.].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape
(1, -1) if it contains a single sample.
```

In [20]:

```
x_test
```

Out[20]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
192	54	1	0	120	188	0	1	113	0	1.4	1	1	3
63	41	1	1	135	203	0	1	132	0	0.0	1	0	1
20	59	1	0	135	234	0	1	161	0	0.5	1	0	3
273	58	1	0	100	234	0	1	156	0	0.1	2	1	3
164	38	1	2	138	175	0	1	173	0	0.0	2	4	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
237	60	1	0	140	293	0	0	170	0	1.2	1	2	3
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
194	60	1	2	140	185	0	0	155	0	3.0	1	0	2
241	59	0	0	174	249	0	1	143	1	0.0	1	0	2
124	39	0	2	94	199	0	1	179	0	0.0	2	0	2

61 rows × 13 columns

In [21]:

```
y_preds = clf.predict(x_test)
y_preds
```

Out[21]:

```
array([0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1], dtype=int64)
```

In [22]:
```
y_test
```

Out[22]:
```
192    0
63     1
20     1
273    0
164    1
      ..
237    0
179    0
194    0
241    0
124    1
Name: target, Length: 61, dtype: int64
```

In [23]:
```
# 4.Evaluate the model on the training data and test data
clf.score(x_train, y_train)
```

Out[23]:
```
1.0
```

In [24]:
```
clf.score(x_test, y_test)
```

Out[24]:
```
0.7540983606557377
```

In [25]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78        37
           1       0.66      0.79      0.72        24

    accuracy                           0.75        61
   macro avg       0.75      0.76      0.75        61
weighted avg       0.77      0.75      0.76        61
```

In [26]:
```
confusion_matrix(y_test, y_preds)
```

Out[26]:
```
array([[27, 10],
       [ 5, 19]], dtype=int64)
```

In [27]:

```
accuracy_score(y_test, y_preds)
```

Out[27]:

```
0.7540983606557377
```

In [28]:

```
# improve a model
# try different amount of n_estimators
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier(n_estimators = i).fit(x_train, y_train)
    print(f"Model accuracy on test set: {clf.score(x_test, y_test) * 100:.2f}%")
    print("")
```

```
Trying model with 10 estimators...
Model accuracy on test set: 75.41%

Trying model with 20 estimators...
Model accuracy on test set: 77.05%

Trying model with 30 estimators...
Model accuracy on test set: 70.49%

Trying model with 40 estimators...
Model accuracy on test set: 72.13%

Trying model with 50 estimators...
Model accuracy on test set: 68.85%

Trying model with 60 estimators...
Model accuracy on test set: 72.13%

Trying model with 70 estimators...
Model accuracy on test set: 72.13%

Trying model with 80 estimators...
Model accuracy on test set: 73.77%

Trying model with 90 estimators...
Model accuracy on test set: 68.85%
```

In [29]:
```
# 6. save a model and Load it
import pickle

pickle.dump(clf, open('random_forest_model_1.pk1', 'wb'))
```

In [30]:
```
loaded_model = pickle.load(open('random_forest_model_1.pk1', 'rb'))
loaded_model.score(x_test, y_test)
```

Out[30]:
```
0.6885245901639344
```

In [31]:
```
# to show the sklearn version we are using
import sklearn
sklearn.show_versions()

# to ignore this warning
import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\USER\anaconda3\lib\site-packages\_distutils_hack\__init__.py:33: UserWarning: Setuptools is rep
lacing distutils.
  warnings.warn("Setuptools is replacing distutils.")


System:
    python: 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]
executable: C:\Users\USER\anaconda3\python.exe
   machine: Windows-10-10.0.19045-SP0

Python dependencies:
          pip: 22.2.2
   setuptools: 63.4.1
      sklearn: 1.0.2
        numpy: 1.21.5
        scipy: 1.9.1
       Cython: 0.29.32
       pandas: 1.4.4
   matplotlib: 3.5.2
       joblib: 1.1.0
threadpoolctl: 2.2.0

Built with OpenMP: True
```

In [32]:
```
# let's listify the contents
what_were_covering = [
'0. An end-to-end scikit-learn workflow',
'1. Getting the data ready',
'2. Choose the right estimator/algorithm for our problems',
'3. Fit the models/algorithm and use it to make predictions on our data',
'4. Evaluating a model',
'5. Improve a model',
'6. Save and load a trained model',
'7. Putting it all together',
]
```

Second part of the lesson

In [33]:
```
what_were_covering
```

Out[33]:
```
['0. An end-to-end scikit-learn workflow',
 '1. Getting the data ready',
 '2. Choose the right estimator/algorithm for our problems',
 '3. Fit the models/algorithm and use it to make predictions on our data',
 '4. Evaluating a model',
 '5. Improve a model',
 '6. Save and load a trained model',
 '7. Putting it all together']
```

In [34]:
```
# standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Getting our data ready to be used with machine learning

Three main things we have to do:

1. Split the data into features and labels (usually 'x' & 'y')
2. Filling (also called inputing) or disregarding missing values
3. Converting non-numerical values to numerical values (also called feature encoding)

In [35]:
```
heart_disease.head()
```

Out[35]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [36]:
```
x = heart_disease.drop('target', axis = 1)
x.head()
```

Out[36]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

In [37]:
```
y = heart_disease['target']
y.head()
```

Out[37]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [38]:
```
# split the data into training and test sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

In [39]:
```
# checking the shape of our data
x_train.shape, x_test.shape, y_train.shape, y_test.shape

# here, test_size applied is = 0.2(20%) that is why roll is 242
```

Out[39]:
```
((242, 13), (61, 13), (242,), (61,))
```

In [40]:
```
# here is the full roll
x.shape
```

Out[40]:
```
(303, 13)
```

In [41]:
```
# here is also the full roll
len(heart_disease)
```

Out[41]:
```
303
```

In [42]:
```
x.shape[0] * 0.8
```

Out[42]:
```
242.4
```

In [43]:
```
242 + 61
```

Out[43]:
```
303
```

make sure the data is all numerical

In [44]:
```
car_sales = pd.read_csv('car-sales-extended.csv')
car_sales.head()
```

Out[44]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431	4	15323
1	BMW	Blue	192714	5	19943
2	Honda	White	84714	4	28343
3	Toyota	White	154365	4	13434
4	Nissan	Blue	181577	3	14043

In [45]:
```
car_sales['Doors'].value_counts()
```

Out[45]:
```
4    856
5     79
3     65
Name: Doors, dtype: int64
```

In [46]:
```
len(car_sales)
```

Out[46]:
```
1000
```

In [47]:
```
car_sales.dtypes
```

Out[47]:
```
Make                object
Colour              object
Odometer (KM)        int64
Doors                int64
Price                int64
dtype: object
```

In [48]:
```
# split the data into 'x' and 'y'
x = car_sales.drop('Price', axis = 1)
y = car_sales['Price']

# split into training and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.2)

```

In [49]:

```
# build machine learning
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6928\2751715000.py in <module>
      2 from sklearn.ensemble import RandomForestRegressor
      3 model = RandomForestRegressor()
----> 4 model.fit(x_train, y_train)
      5 model.score(x_test, y_test)

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in fit(self, X, y, sample_weight)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is not supported.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    329         )

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately,
**check_params)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_X_y(X, y, accept_sparse, accept_large
_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, en
sure_min_features, y_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963 
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_la
rge_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_fe
atures, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\generic.py in __array__(self, dtype)
   2062 
```

```
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065 
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'Toyota'
```

In [50]:

```
# to know what x is before convrsion
x.head()
```

Out[50]:

	Make	Colour	Odometer (KM)	Doors
0	Honda	White	35431	4
1	BMW	Blue	192714	5
2	Honda	White	84714	4
3	Toyota	White	154365	4
4	Nissan	Blue	181577	3

In [51]:

```python
# to convert the strings/objects to integer(numbers)(make and colour) from the data set to solve the
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# find the category features to convert to numbers
categorical_features = ['Make', 'Colour', 'Doors']
one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot',
                                  one_hot,
                                  categorical_features)],
                                remainder = 'passthrough')

transformed_x = transformer.fit_transform(x)
transformed_x
```

Out[51]:

```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 3.54310e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        1.00000e+00, 1.92714e+05],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 8.47140e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 6.66040e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.15883e+05],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        0.00000e+00, 2.48360e+05]])
```

In [52]:
```
# putting the above data in a data frame

pd.DataFrame(transformed_x).head()
```

Out[52]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	35431.0
1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	192714.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	84714.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	154365.0
4	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	181577.0

In [53]:
```
# another way to transform our column to integers(numbers)

dummies = pd.get_dummies(car_sales[['Make', 'Colour', 'Doors']])
dummies.head()
```

Out[53]:

	Doors	Make_BMW	Make_Honda	Make_Nissan	Make_Toyota	Colour_Black	Colour_Blue	Colour_Green	Colour_Red	Colour_White
0	4	0	1	0	0	0	0	0	0	1
1	5	1	0	0	0	0	1	0	0	0
2	4	0	1	0	0	0	0	0	0	1
3	4	0	0	0	1	0	0	0	0	1
4	3	0	0	1	0	0	1	0	0	0

In [54]:
```
# since our data has turn to zero and ones
# let's refill the model

np.random.seed(42)
x_train, x_test, y_train, y_test = train_test_split(transformed_x,
                                                    y,
                                                    test_size=0.2)
model.fit(x_train, y_train)
```

Out[54]:
```
RandomForestRegressor()
```

In [55]:
```
model.score(x_test, y_test)
```

Out[55]:
```
0.3235867221569877
```

1.2 what if there were missing values?

1. Fill them with some value (also known as imputation).
2. Remove the samples with missing data altogether.

In [56]:
```
# import car sales missind data
car_sales_missing = pd.read_csv('car-sales-extended-missing-data.csv')
car_sales_missing.head()
```

Out[56]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0

In [57]:
```
# to check missing data
car_sales_missing.isna().sum()
```

Out[57]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [58]:
```
# create x & y
x = car_sales_missing.drop('Price', axis = 1)
y = car_sales_missing['Price']
```

In [59]:

```
# Let's try and convert our data to numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# find the category features to convert to numbers
categorical_features = ['Make', 'Colour', 'Doors']
one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot',
                                  one_hot,
                                  categorical_features)],
                                remainder = 'passthrough')

transformed_x = transformer.fit_transform(x)
transformed_x
```

Out[59]:

```
<1000x16 sparse matrix of type '<class 'numpy.float64'>'
        with 4000 stored elements in Compressed Sparse Row format>
```

In [60]:

```
car_sales_missing
```

Out[60]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [61]:
```
# above worked directly without filling up with pandas
```

In [62]:
```
car_sales_missing['Doors'].value_counts()
```

Out[62]:
```
4.0    811
5.0     75
3.0     64
Name: Doors, dtype: int64
```

option 1: fill missing data with pandas

In [63]:
```
# fill the 'Make' column
car_sales_missing['Make'].fillna('missing', inplace=True)

# fill the 'Colour' column
car_sales_missing['Colour'].fillna('missing', inplace = True)

# fill the 'Odometer (KM)' column
car_sales_missing['Odometer (KM)'].fillna(car_sales_missing['Odometer (KM)'].mean(), inplace = True)

# fill the 'Doors' column
car_sales_missing['Doors'].fillna(4, inplace = True)
```

In [64]:
```
# check our dataframe again
car_sales_missing.isna().sum()
```

Out[64]:
```
Make              0
Colour            0
Odometer (KM)     0
Doors             0
Price            50
dtype: int64
```

In [65]:
```
# remove rows with missing price values
car_sales_missing.dropna(inplace = True)

```

In [66]:
```
car_sales_missing.isna().sum()
```

Out[66]:
```
Make             0
Colour           0
Odometer (KM)    0
Doors            0
Price            0
dtype: int64
```

In [67]:
```
len(car_sales_missing)
```

Out[67]:
```
950
```

In [68]:
```
x = car_sales_missing.drop('Price', axis = 1)
y = car_sales_missing['Price']
```

In [69]:
```python
# let's bring back our code now

# let's try and convert our data to numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# find the category features to convert to numbers
categorical_features = ['Make', 'Colour', 'Doors']
one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot',
                                  one_hot,
                                  categorical_features)],
                                remainder = 'passthrough')

transformed_x = transformer.fit_transform(car_sales_missing)
transformed_x
```

Out[69]:
```
array([[0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        3.54310e+04, 1.53230e+04],
       [1.00000e+00, 0.00000e+00, 0.00000e+00, ..., 1.00000e+00,
        1.92714e+05, 1.99430e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        8.47140e+04, 2.83430e+04],
       ...,
       [0.00000e+00, 0.00000e+00, 1.00000e+00, ..., 0.00000e+00,
        6.66040e+04, 3.15700e+04],
       [0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.15883e+05, 4.00100e+03],
       [0.00000e+00, 0.00000e+00, 0.00000e+00, ..., 0.00000e+00,
        2.48360e+05, 1.27320e+04]])
```

In [70]:
```
# putting above data into a dataframe
pd.DataFrame(transformed_x).head()
```

Out[70]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	35431.0	15323.0
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	192714.0	19943.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	84714.0	28343.0
3	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	154365.0	13434.0
4	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	181577.0	14043.0

Option 2: fill missing value with scikit-learn

In [71]:
```
car_sales_missing = pd.read_csv('car-sales-extended-missing-data.csv')
car_sales_missing.head()
```

Out[71]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0

In [72]:
```
# to check for missing data
car_sales_missing.isna().sum()
```

Out[72]:
```
Make             49
Colour           50
Odometer (KM)    50
Doors            50
Price            50
dtype: int64
```

In [73]:
```python
car_sales_missing.dropna(subset = ['Price'], inplace = True)
car_sales_missing
```

Out[73]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

950 rows × 5 columns

In [74]:
```python
car_sales_missing.isna().sum()
```

Out[74]:
```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
Price             0
dtype: int64
```

In [75]:
```python
# split into x and y
x = car_sales_missing.drop('Price', axis = 1)
y = car_sales_missing['Price']
```

In [76]:

```
x.isna().sum()
```

Out[76]:

```
Make             47
Colour           46
Odometer (KM)    48
Doors            47
dtype: int64
```

In [77]:

```
# to fix missing data/values with sklearn

from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

# fill categorical values 'missing' & numerical values with 'mean'
cat_imputer = SimpleImputer(strategy = 'constant', fill_value = 'missing')
door_imputer = SimpleImputer(strategy = 'constant', fill_value = 4)
num_imputer = SimpleImputer(strategy = 'mean')

# Define colums
cat_features = ['Make', 'Colour']
door_feature = ['Doors']
num_features = ['Odometer (KM)']

# create an inputer (sosmething that fills missing data)
imputer = ColumnTransformer([
    ('cat_imputer', cat_imputer, cat_features),
    ('door_imputer', door_imputer, door_feature),
    ('num_imputer', num_imputer, num_features)
])

# transform the data
filled_x = imputer.fit_transform(x)
filled_x

```

Out[77]:

```
array([['Honda', 'White', 4.0, 35431.0],
       ['BMW', 'Blue', 5.0, 192714.0],
       ['Honda', 'White', 4.0, 84714.0],
       ...,
       ['Nissan', 'Blue', 4.0, 66604.0],
       ['Honda', 'White', 4.0, 215883.0],
       ['Toyota', 'Blue', 4.0, 248360.0]], dtype=object)
```

In [78]:
```
car_sales_filled = pd.DataFrame(filled_x,
                                columns = ['Make', 'Colour', 'Doors', 'Odometer (KM)'])
car_sales_filled.head()
```

Out[78]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4.0	35431.0
1	BMW	Blue	5.0	192714.0
2	Honda	White	4.0	84714.0
3	Toyota	White	4.0	154365.0
4	Nissan	Blue	3.0	181577.0

In [79]:
```
car_sales_filled.isna().sum()
```

Out[79]:
```
Make             0
Colour           0
Doors            0
Odometer (KM)    0
dtype: int64
```

In [80]:
```
car_sales_filled.head()
```

Out[80]:

	Make	Colour	Doors	Odometer (KM)
0	Honda	White	4.0	35431.0
1	BMW	Blue	5.0	192714.0
2	Honda	White	4.0	84714.0
3	Toyota	White	4.0	154365.0
4	Nissan	Blue	3.0	181577.0

In [81]:
```
# let's try and convert our data to numbers
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# find the category features to convert to numbers
categorical_features = ['Make', 'Colour', 'Doors']
one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot',
                                  one_hot,
                                  categorical_features)],
                                remainder = 'passthrough')

transformed_x = transformer.fit_transform(car_sales_filled)
transformed_x
```

Out[81]:
```
<950x15 sparse matrix of type '<class 'numpy.float64'>'
        with 3800 stored elements in Compressed Sparse Row format>
```

In [82]:
```
# Now we've got our data as numbers and filled (no missing values)
# let's fit a model
np.random.seed(42)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(transformed_x,
                                                    y,
                                                     test_size = 0.2)
model = RandomForestRegressor()
model.fit(x_train, y_train)
model.score(x_test, y_test)

```

Out[82]:
```
0.21990196728583944
```

In [83]:
```
# the above perfomed worse cos it hs 950 samples
len(car_sales_filled), len(car_sales)
```

Out[83]:
```
(950, 1000)
```

2. Choosing the right estimator/algorithm for our problem

scikit-learn uses estimator as another term for machine learning model or algorithm.

- Classification - predicting whether a sample is one thing or another.
- Regression - predicting a number

step 1 - check the scikit-learn machine learning mapp... [https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

2.1 picking a machine learning model for a regression problem

In [84]:
```
# import Boston housing dataset
from sklearn.datasets import load_boston
boston = load_boston()
boston;
```

In [85]:
```
boston_df = pd.DataFrame(boston['data'], columns = boston['feature_names'])
boston_df['target'] = pd.Series(boston['target'])
boston_df.head()
```

Out[85]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [86]:
```
# how many samples?
len(boston_df)
```

Out[86]: 506

In [87]:

```
# let try the Ridge Regression model
from sklearn.linear_model import Ridge

# setup random seed
np.random.seed(42)

# create the data
x = boston_df.drop('target', axis = 1)
y = boston_df['target']

# split into train and test sets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)

# instantiate Ridge model
model = Ridge()
model.fit(x_train, y_train)

# checck the score of the Ridge
model.score(x_test, y_test)
```

Out[87]:

```
0.6662221670168518
```

How do we improve this score?

What if Ridge wasn't working

let's refer back to the map... [https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

In [88]:
```
# Let's try the Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

# set random seed
np.random.seed(42)

# create the data
x = boston_df.drop('target', axis = 1)
y = boston_df['target']

# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate random forest regressor
rf = RandomForestRegressor()
rf.fit(x_train, y_train)

# evaluate the random forest regressor
rf.score(x_test, y_test)

```

Out[88]:
```
0.8654448653350507
```

In [89]:
```
# ccheck the Ridge model again
model.score(x_test, y_test)

```

Out[89]:
```
0.6662221670168518
```

Choosing an estimator for a classification problem

let's go tp the maap...https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html (https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

In [90]:

```
heart_disease = pd.read_csv('heart-disease.csv')
heart_disease.head()
```

Out[90]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [91]:

```
len(heart_disease)
```

Out[91]:

```
303
```

Consulting the map and it says to try linearSVC.

In [92]:
```
# import the linearSVC estimator class
from sklearn.svm import LinearSVC

# set up random seed
np.random.seed(42)

# make the data
x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate LinearSVC
ln = LinearSVC()
ln.fit(x_train, y_train)

#Evaluate the LinearSVC
ln.score(x_test, y_test)

```

Out[92]:
```
0.8688524590163934
```

In [93]:
```
heart_disease['target'].value_counts()
```

Out[93]:
```
1    165
0    138
Name: target, dtype: int64
```

In [94]:
```
# import the RandomForestClassifier estimator class
from sklearn.ensemble import RandomForestClassifier

# set up random seed
np.random.seed(42)

# make the data
x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)

#Evaluate the RandomForestClassifier
rfc.score(x_test, y_test)

```

Out[94]:
```
0.8524590163934426
```

Tidbit:

1. if you have structured(tables) data, use ensemble methods
2. if you have unstructured(images, audios etc) data, use deep learning or transfer learning

In [95]:
```
heart_disease.head()  # this is a structured data
```

Out[95]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

3. Fit the model/algorithm on our data and use it to make predictions

3.1 Fitting the model to the data

Different names for:

- x = features, features variables, data
- y = labels, targets, target variables

In [96]:
```
# import the RandomForestClassifier estimator class
from sklearn.ensemble import RandomForestClassifier

# set up random seed
np.random.seed(42)

# make the data
x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate RandomForestClassifier
rfc = RandomForestClassifier()

# fit the model to the data (training the machine learning model)
rfc.fit(x_train, y_train)

#Evaluate the RandomForestClassifier (use the patterns the model has learned)
rfc.score(x_test, y_test)
```

Out[96]:
```
0.8524590163934426
```

In [97]:
```
x.head()
```

Out[97]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2

In [98]:
```
y.head()
```

Out[98]:
```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

In [99]:
```
y.tail()
```

Out[99]:
```
298    0
299    0
300    0
301    0
302    0
Name: target, dtype: int64
```

3.2 make predictions using a machine learning model

Two ways to make predictions

1. predict()
2. predict_proba()

In [100]:
```
# use a trained model to make predictions

rfc.predict(np.array([1, 7, 8, 3, 4]))  # this doesn't work..
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6928\4195040950.py in <module>
      1 # use a trained model to make predictions
      2 
----> 3 rfc.predict(np.array([1, 7, 8, 3, 4]))  # this doesn't work..

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    806             The predicted classes.
    807         """
--> 808         proba = self.predict_proba(X)
    809 
    810         if self.n_outputs_ == 1:

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict_proba(self, X)
    848         check_is_fitted(self)
    849         # Check data
--> 850         X = self._validate_X_predict(X)
    851 
    852         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    577         Validate X whenever one tries to predict, apply, predict_proba."""
    578         check_is_fitted(self)
--> 579         X = self._validate_data(X, dtype=DTYPE, accept_sparse="csr", reset=False)
    580         if issparse(X) and (X.indices.dtype != np.intc or X.indptr.dtype != np.intc):
    581             raise ValueError("No support for np.int64 index based sparse matrices")

~\anaconda3\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, reset, validate_separately,
**check_params)
    564             raise ValueError("Validation should be done on X, y or both.")
    565         elif not no_val_X and no_val_y:
--> 566             X = check_array(X, **check_params)
    567             out = X
    568         elif no_val_X and not no_val_y:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_la
rge_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_fe
atures, estimator)
    767             # If input is 1D raise error
    768             if array.ndim == 1:
--> 769                 raise ValueError(
    770                     "Expected 2D array, got 1D array instead:\narray={}.\n"
```

```
    771                     "Reshape your data either using array.reshape(-1, 1) if "

ValueError: Expected 2D array, got 1D array instead:
array=[1. 7. 8. 3. 4.].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape
(1, -1) if it contains a single sample.
```

In [101]:
```
1 x_test.head()
```

Out[101]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

In [102]:
```
1 rfc.predict(x_test)
```

Out[102]:
```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [103]:
```
1 y_test
```

Out[103]:
```
179    0
228    0
111    1
246    0
60     1
      ..
249    0
104    1
300    0
193    0
184    0
Name: target, Length: 61, dtype: int64
```

In [104]:
```
# we can put y_test in form of x-test array

np.array(y_test)
```

Out[104]:
```
array([0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

In [105]:
```
# compare predictions to truth labels to evaluate the model
y_preds = rfc.predict(x_test)
np.mean(y_preds == y_test)
```

Out[105]:
```
0.8524590163934426
```

In [106]:
```
rfc.score(x_test, y_test)
```

Out[106]:
```
0.8524590163934426
```

In [107]:
```
# another way of doing the above
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_preds)
```

Out[107]:
```
0.8524590163934426
```

Make predictions with predict_proba()

In [108]:
```
# Predicts_proba() returns probabilities of a classification label
rfc.predict_proba(x_test[: 5])

```

Out[108]:
```
array([[0.89, 0.11],
       [0.49, 0.51],
       [0.43, 0.57],
       [0.84, 0.16],
       [0.18, 0.82]])
```

In [109]:
```
# let's predict() on the same data...
rfc.predict(x_test [: 5])
```

Out[109]:
```
array([0, 1, 1, 0, 1], dtype=int64)
```

In [110]:
```
x_test[: 5]
```

Out[110]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
179	57	1	0	150	276	0	0	112	1	0.6	1	1	1
228	59	1	3	170	288	0	0	159	0	0.2	1	0	3
111	57	1	2	150	126	1	1	173	0	0.2	2	1	3
246	56	0	0	134	409	0	0	150	1	1.9	1	2	3
60	71	0	2	110	265	1	0	130	0	0.0	2	1	2

In [111]:
```
heart_disease['target'].value_counts()
```

Out[111]:
```
1    165
0    138
Name: target, dtype: int64
```

making prediction with Regression Model

predict() can also be used for regression models

In [112]:
```
boston_df
```

Out[112]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [113]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# create the data
x = boston_df.drop('target', axis = 1)
y = boston_df['target']

# split into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate and fit the model
model = RandomForestRegressor().fit(x_train, y_train)

# make predictions

y_preds = model.predict(x_test)
```

In [114]:
```
y_preds[:10]
```

Out[114]:
```
array([23.081, 30.574, 16.759, 23.46 , 16.893, 21.644, 19.113, 15.334,
       21.14 , 20.639])
```

In [115]:
```
np.array(y_test[:10])
```

Out[115]:
```
array([23.6, 32.4, 13.6, 22.8, 16.1, 20. , 17.8, 14. , 19.6, 16.8])
```

In [116]:
```
# compare the predictions to the truth
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_preds)
```

Out[116]:
```
2.136382352941176
```

4. Evaluating a machine learning model(score)

Three ways to evaluate scikit-learn models/estimators:

1. Estimator score method
2. The scoring parameter
3. Problem specific metric functions

4.1 Evaluating a model with the 'score method'

In [117]:
```
heart_disease.head()
```

Out[117]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [118]:
```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

rfc = RandomForestClassifier()

rfc.fit(x_train, y_train)
```

Out[118]: RandomForestClassifier()

In [119]:
```
rfc.score(x_train, y_train)
```

Out[119]: 1.0

In [120]:
```
rfc.score(x_test, y_test)
```

Out[120]: 0.8524590163934426

let's do the same but for regression...

In [121]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

# create the data
x = boston_df.drop('target', axis = 1)
y = boston_df['target']

# split into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# instantiate and fit the model
model = RandomForestRegressor().fit(x_train, y_train)

```

In [122]:
```
model.score(x_test, y_test)
```

Out[122]: 0.8654448653350507

In [123]:
```
model.score(x_train, y_train)
```

Out[123]: 0.9763520974033731

4.2 Evaluating a model using scoring parameters (cross validation)

In [124]:
```
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

rfc = RandomForestClassifier()

rfc.fit(x_train, y_train);

```

In [125]:
```
rfc.score(x_train, y_train)
```

Out[125]:
```
1.0
```

In [126]:
```
rfc.score(x_test, y_test)
```

Out[126]:
```
0.8524590163934426
```

In [127]:
```
cross_val_score(rfc, x, y)
```

Out[127]:
```
array([0.81967213, 0.86885246, 0.81967213, 0.78333333, 0.76666667])
```

In [128]:
```
cross_val_score(rfc, x, y, cv = 10)
```

Out[128]:
```
array([0.90322581, 0.80645161, 0.87096774, 0.9       , 0.86666667,
       0.8       , 0.73333333, 0.86666667, 0.73333333, 0.8       ])
```

In [129]:
```
np.random.seed(42)

# single training and test split score
rfc_single_score = rfc.score(x_test, y_test)

# take the mean of 5-fold cross validation score
rfc_cross_val_score = np.mean(cross_val_score(rfc, x, y))

# compare the two
rfc_single_score, rfc_cross_val_score
```

Out[129]:
```
(0.8524590163934426, 0.8248087431693989)
```

In [130]:
```
# default scoring paramter of classifier = mean accuracy
rfc.score(x, y)
```

Out[130]:
```
0.9702970297029703
```

In [131]:
```
# scoring parameter set to None by defaault
cross_val_score(rfc, x, y, scoring = None)
```

Out[131]:
```
array([0.78688525, 0.86885246, 0.80327869, 0.78333333, 0.76666667])
```

In [132]:
```
rfc.score(x_train, y_train)
```

Out[132]:
```
1.0
```

4.2.1 Classification model evaluation metrics

1. Accuracy
2. Area under ROC curve
3. confusion matrix
4. classification report

Accuracy

In [133]:
```
heart_disease.head()
```

Out[133]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [134]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
cross_val_score = cross_val_score(clf, x, y, cv = 5)
```

In [135]:
```
np.mean(cross_val_score)
```

Out[135]: 0.811639344262295

In [136]:
```
print(f'Heart Disease Classifier Cross-Validated Acurracy: {np.mean(cross_val_score) * 100: .2f}%')
```

```
Heart Disease Classifier Cross-Validated Acurracy:  81.16%
```

Area under the reciever operating characteristics curve (AUC/ROC)

- Area under curve (AUC)
- ROC curve

ROC curves are a comparison of a model's true positive (tpr) versus a model's false positive rate (fpr)

- True positive = model predicts 1 when truth is 1
- False positive = model predicts 1 when truth is 0
- True negative = models predicts 0 when truth is 0
- False negative = model predicts 0 when truth is 1

In [137]:
```
from sklearn.metrics import roc_curve

# make predictions with probabilities

y_probs = clf.predict_proba(x_test)

y_probs[: 10], len(y_probs)
```

Out[137]:
```
(array([[0.89, 0.11],
        [0.49, 0.51],
        [0.43, 0.57],
        [0.84, 0.16],
        [0.18, 0.82],
        [0.14, 0.86],
        [0.36, 0.64],
        [0.95, 0.05],
        [0.99, 0.01],
        [0.47, 0.53]]),
 61)
```

In [138]:
```
y_probs_positive = y_probs[:, 1]
y_probs_positive[: 10]
```

Out[138]:
```
array([0.11, 0.51, 0.57, 0.16, 0.82, 0.86, 0.64, 0.05, 0.01, 0.53])
```

In [139]:
```
y_probs_positives = y_probs[:, 0]
y_probs_positive[:10]
```

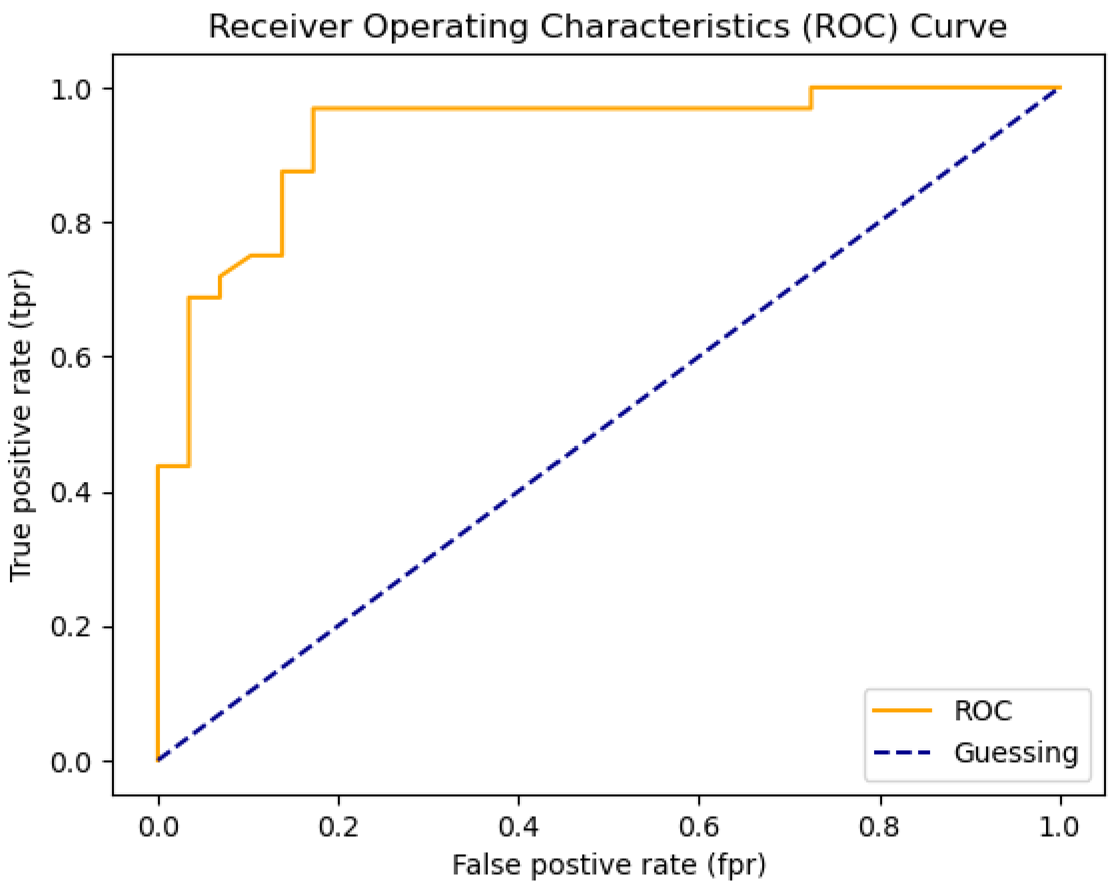
Out[139]:
```
array([0.11, 0.51, 0.57, 0.16, 0.82, 0.86, 0.64, 0.05, 0.01, 0.53])
```

In [140]:
```
# calculate fpr, tpr and thresholds

fpr, tpr, thresholds = roc_curve(y_test, y_probs_positive)

# check the false postive
fpr

```

Out[140]:
```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.03448276, 0.03448276, 0.03448276, 0.03448276, 0.06896552,
       0.06896552, 0.10344828, 0.13793103, 0.13793103, 0.17241379,
       0.17241379, 0.27586207, 0.4137931 , 0.48275862, 0.55172414,
       0.65517241, 0.72413793, 0.72413793, 0.82758621, 1.        ])
```

In [141]:

```
# create a function for plotting ROC curves
import matplotlib.pyplot as plt

def plot_roc_curve(fpr, tpr):

    """
    plots a Roc curve given the false postive rate (frp)
    and true positive rate (tpr) of a model.
    """
    # plot roc curves
    plt.plot(fpr, tpr, color = 'orange', label = 'ROC')

    #plot line with no predictive power (baseline)
    plt.plot([0, 1], [0, 1], color = 'darkblue', linestyle = '--', label = 'Guessing')

    #customize the plot
    plt.xlabel('False postive rate (fpr)')
    plt.ylabel('True positive rate (tpr)')
    plt.title('Receiver Operating Characteristics (ROC) Curve')
    plt.legend()
    plt.show()

plot_roc_curve(fpr, tpr)
```

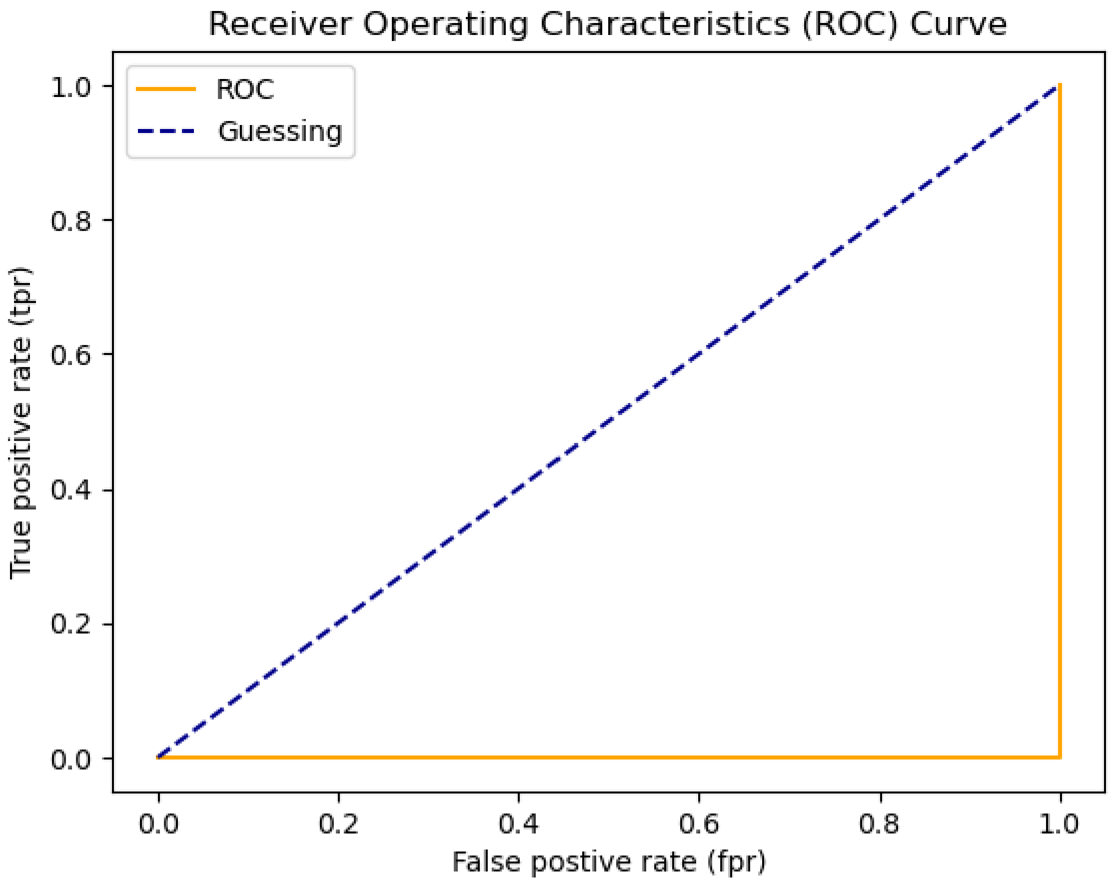


In [142]:

```
# 'let's look at the AUC score

from sklearn.metrics import roc_auc_score

roc_auc_score(y_test, y_probs_positive)
```

Out[142]: 0.9304956896551724

In [143]:
```
# plot perfect ROC curve and AUC score

fpr, tpr, thresholds = roc_curve(y_test, y_test)
plot_roc_curve(tpr, fpr)
```



In [144]:
```
# perfect AUC score
roc_auc_score(y_test, y_test)
```

Out[144]: 1.0

Confusion matrix

A confusion matrix is a quick way to compare the label a model predicts and the actual labels it was supposed to predict.

In essense, giving you an idea of where the model is getting confused

In [145]:
```
from sklearn.metrics import confusion_matrix

y_preds = clf.predict(x_test)
confusion_matrix(y_test, y_preds)
```

Out[145]:
```
array([[24,  5],
       [ 4, 28]], dtype=int64)
```

In [146]:
```
# visualise confusion matrix with pd.crosstab()

pd.crosstab(y_test,
            y_preds,
            rownames = ['Actual Labels'],
             colnames = ['Predicted Labels'])
```

Out[146]:

Predicted Labels	**0**	**1**
Actual Labels		
0	24	5
1	4	28

In [147]:
```
24 + 5 + 4 + 28
```

Out[147]: 61

In [148]:
```
len(y_preds)
```

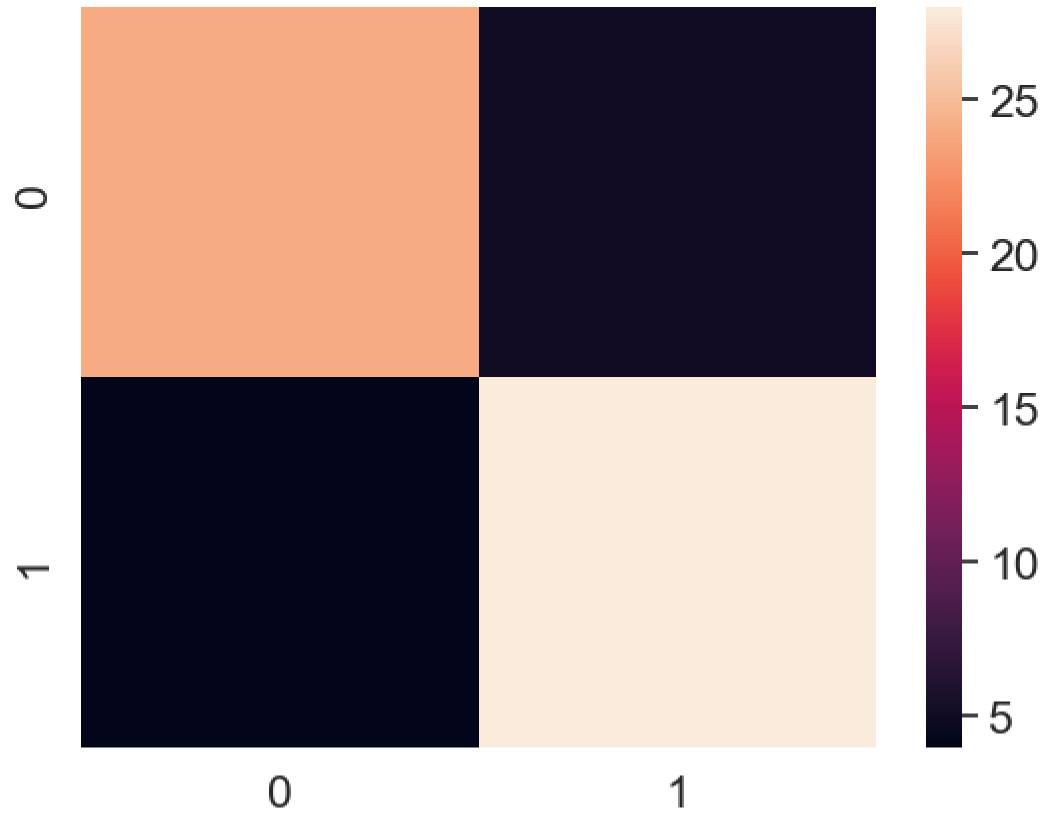
Out[148]: 61

In [149]:
```
len(y_test)
```

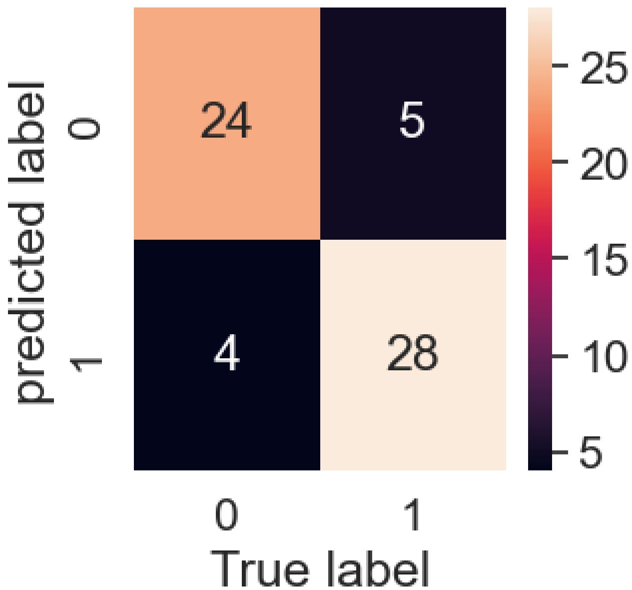
Out[149]: 61

In [150]:

```
# make confusion matrix more visual with seaborn's heatmap
import seaborn as sns

# set the font scale
sns.set(font_scale = 1.5 )

# create a confusion matrix
conf_mat = confusion_matrix(y_test, y_preds)

# plot it using seaborn
sns.heatmap(conf_mat);
```



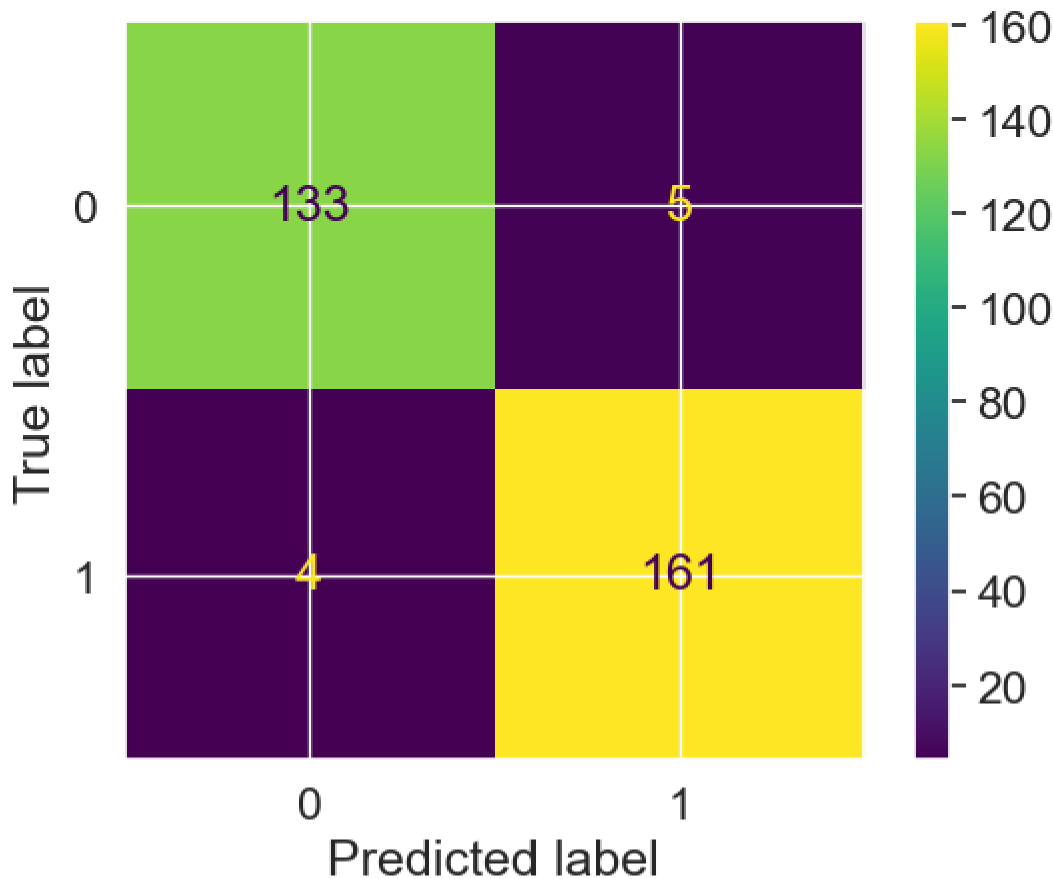
In [151]:
```
def plot_conf_mat(conf_mat):
    """
    plot a confusion matrix using seaborn's heatmap()
    """
    fig, ax = plt.subplots(figsize = (3, 3 ))
    ax = sns.heatmap(conf_mat,
                     annot = True, # Annotes the boxes with conf_mat info
                     cbar = True)
    plt.xlabel('True label')
    plt.ylabel('predicted label');

plot_conf_mat(conf_mat)
```



note, if the seabon heatmap has broekn notations u can fix it with this code:

```
*bottom, top = ax.get_ylim()
*ax.set_ylim(bottom + 0.5, top -0.5)
```

In [152]:
```python
# we can do the above using sklearn instead of seaborn(sns)

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, x, y);
```



Classification report

In [153]:
```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.86      0.83      0.84        29
           1       0.85      0.88      0.86        32

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

In [154]:
```python
# Where precision and recall become valuable
disease_true = np.zeros(10000)
disease_true[0] = 1 # only one positive case

disease_preds = np.zeros(10000) # model predicts every case as 0

pd.DataFrame(classification_report(disease_true,
                                   disease_preds,
                                   output_dict = True))
```

Out[154]:

	0.0	1.0	accuracy	macro avg	weighted avg
precision	0.99990	0.0	0.9999	0.499950	0.99980
recall	1.00000	0.0	0.9999	0.500000	0.99990
f1-score	0.99995	0.0	0.9999	0.499975	0.99985
support	9999.00000	1.0	0.9999	10000.000000	10000.00000

To summarise classification metrics:

- **Accuracy** is a good measure to start with if all classes are balanced(e.g same amount of samples which are labelled with 0 or 1).
- **Precision** and **recall** become more important when classes are imbalanced
- if false positive predictions are worse than false negatives, aim for higher precision
- if false negative predicts are worse than false positives, aim for higher recall

- **F1-score** is a combination of precision and recall

4.2.2 Regression model evaluation metrics

1. R^2 (pronounced r-squared) or coefficient of determination
2. mean absolute error(MAE)
3. mean squared error(MSE)

R^2

what R-squared does: Compares your models predictions to the mean of the targets. Values can range from negative infinity (a very poor model) to 1. For example, if all your model does is predict the mean of the targets, it's R^2 value would be 0. And if your model perfectly predicts a range of numbers it's R^2 value would be 1.

In [155]:
```
boston_df.head()
```

Out[155]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [156]:
```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

x = boston_df.drop('target', axis = 1)
y = boston_df['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

model = RandomForestRegressor()
model.fit(x_train, y_train);
```

In [157]:
```
model.score(x_test, y_test)
```

Out[157]:
```
0.8654448653350507
```

In [158]:
```
from sklearn.metrics import r2_score

# Fill an array with y_test mean

y_test_mean = np.full(len(y_test), y_test.mean())
```

In [159]:
```
y_test.mean()
```

Out[159]:
```
21.488235294117654
```

In [160]:
```
r2_score(y_test, y_test_mean)
```

Out[160]:
```
2.220446049250313e-16
```

In [161]:
```
r2_score(y_test, y_test)
```

Out[161]:
```
1.0
```

Mean absolute error (MAE)

MAE is the average of the absolute differences between predictions and actual values. It gives you an idea of how wrong your models predictions are.

In [162]:
```python
# mean absolute error
from sklearn.metrics import mean_absolute_error

y_preds = model.predict(x_test)
mae = mean_absolute_error(y_test, y_preds)
mae
```

Out[162]: 2.136382352941176

In [163]:
```python
df = pd.DataFrame(data={'actual values': y_test, 'predicted values': y_preds})
df['differences'] = df['predicted values'] - df['actual values']
df
```

Out[163]:

	actual values	predicted values	differences
173	23.6	23.081	-0.519
274	32.4	30.574	-1.826
491	13.6	16.759	3.159
72	22.8	23.460	0.660
452	16.1	16.893	0.793
...	...	...	...
412	17.9	13.159	-4.741
436	9.6	12.476	2.876
411	17.2	13.612	-3.588
86	22.5	20.205	-2.295
75	21.4	23.832	2.432

102 rows × 3 columns

Mean squared error (MSE)

In [164]:
```
# mean squared error
from sklearn.metrics import mean_squared_error

y_preds = model.predict(x_test)
mse = mean_squared_error(y_test, y_preds)
mse
```

Out[164]:
```
9.867437068627442
```

In [165]:
```
# let's calculate MSE by hand
squared = np.square(df['differences'])
squared.mean()
```

Out[165]:
```
9.867437068627439
```

4.2.3 Finally using the scoring parameter

In [166]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

x_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
model = RandomForestClassifier()
```

In [167]:
```
np.random.seed(42)
cv_acc = cross_val_score(model, x, y)
cv_acc
```

Out[167]:
```
array([0.81967213, 0.90163934, 0.83606557, 0.78333333, 0.78333333])
```

In [168]:
```
# cross-validated accuracy
print(f'The cross-validated accuracy is: {np.mean(cv_acc) * 100: .2f}%')
```

```
The cross-validated accuracy is:  82.48%
```

In [169]:
```
# this will be the same value with the above code
# here we use 'acurracy'
np.random.seed(42)
cv_acc = cross_val_score(model, x, y, scoring = 'accuracy')
print(f'The cross-validated accuracy is: {np.mean(cv_acc) * 100: .2f}%')
```

```
The cross-validated accuracy is:  82.48%
```

In [170]:
```
# use of 'precision'

cv_precision = cross_val_score(model, x, y, scoring = 'precision')
cv_precision
```

Out[170]:
```
array([0.76315789, 0.90322581, 0.83870968, 0.79411765, 0.74358974])
```

In [171]:
```
np.mean(cv_precision )
```

Out[171]:
```
0.8085601538512754
```

In [172]:
```
# use of Recall
cv_recall = cross_val_score(model, x, y, scoring = 'recall')
np.mean(cv_recall)
```

Out[172]:
```
0.8424242424242424
```

In [173]:
```
# use of f1-score
cv_f1 = cross_val_score(model, x, y, scoring ='f1')
np.mean(cv_f1)
```

Out[173]:
```
0.841476533416832
```

How about regression model

In [174]:
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor

np.random.seed(42)

x = boston_df.drop('target', axis = 1)
y = boston_df['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
rfr = RandomForestRegressor()
```

In [175]:
```
np.random.seed(42)
cv_r2 = cross_val_score(rfr, x, y, scoring = None)
cv_r2
```

Out[175]:
```
array([0.77231143, 0.86035935, 0.74664002, 0.47632078, 0.26630379])
```

In [176]:
```
np.mean(cv_r2)
```

Out[176]:
```
0.6243870737930857
```

In [177]:
```
np.random.seed(42)
cv_r2 = cross_val_score(rfr, x, y, scoring = 'r2')
cv_r2
```

Out[177]:
```
array([0.77231143, 0.86035935, 0.74664002, 0.47632078, 0.26630379])
```

In [178]:
```
# mean absolute error
cv_mae = cross_val_score(rfr, x, y, scoring = 'neg_mean_absolute_error')
cv_mae
```

Out[178]:
```
array([-2.13045098, -2.49771287, -3.45471287, -3.81509901, -3.11813861])
```

In [179]:
```
# mean squared error

cv_mse = cross_val_score(rfr, x, y, scoring = 'neg_mean_squared_error')
cv_mse
```

Out[179]:
```
array([ -7.8141513 , -12.94343325, -19.11614042, -46.28783248,
       -19.48161818])
```

In [180]:
```
np.mean(cv_mse)
```

Out[180]: -21.12863512415064

4.3 using different evaluation metrics as scikit-learn functions.

classification evaluation functions

In [181]:
```
heart_disease.head(2)
```

Out[181]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1

In [182]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

np.random.seed(42)

x = heart_disease.drop('target', axis = 1)
y = heart_disease['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
y_preds = clf.predict(x_test)

# evaluate the classifier

print('Classifier metrics on the test set')
print(f'Accuracy: {accuracy_score(y_test, y_preds) * 100: .2f}%')
print(f'Precision: {precision_score(y_test, y_preds)}')
print(f'Recall: {recall_score(y_test, y_preds)}')
print(f'F1: {f1_score(y_test, y_preds)}')
```

```
Classifier metrics on the test set
Accuracy:  85.25%
Precision: 0.8484848484848485
Recall: 0.875
F1: 0.8615384615384615
```

Regression evaluation functions

In [183]:

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

np.random.seed(42)

x = boston_df.drop('target', axis = 1)
y = boston_df['target']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
model = RandomForestRegressor()
model.fit(x_train, y_train)
y_preds = model.predict(x_test)

# Evaluate the regression model
print('Regression model metrics on the test set')
print(f'R2: {r2_score(y_test, y_preds)* 100:.2f}%')
print(f'MAE: {mean_absolute_error(y_test, y_preds)}')
print(f'MSA: {mean_squared_error(y_test, y_preds)}')
```

```
Regression model metrics on the test set
R2: 86.54%
MAE: 2.136382352941176
MSA: 9.867437068627442
```

In [184]:

```
what_were_covering
```

Out[184]:

```
['0. An end-to-end scikit-learn workflow',
 '1. Getting the data ready',
 '2. Choose the right estimator/algorithm for our problems',
 '3. Fit the models/algorithm and use it to make predictions on our data',
 '4. Evaluating a model',
 '5. Improve a model',
 '6. Save and load a trained model',
 '7. Putting it all together']
```

5. Improving a model

First predictions = baseline predictions First model = baseline model

from a data perspective:

- could we collect more data? (generally, the more data, the better)
- could we improve our data?

From a model perspective:

- is there a better model we could use?
- could we improve the current model?

Hyperparameters vs Parameters

- parameters = model find these patterns in data
- Hyperparameters = settings on a model you can adjust to (potentially) improve its ability to find patterns

Three ways ro adjust hperparameters:

1. By hand
2. Random with RandomSearchCV
3. Exhaustively with GridsearchCV

In [185]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
```

In [186]:
```
# these are different hyperparameters we can adjust in RandomForestClassifier
clf.get_params()
```

Out[186]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

5.1 Tuning Hyperparameters by hand

let's make 3 sets, training, validation and test.

In [187]:
```
clf.get_params()
```

Out[187]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

we are going to try adjust:

- max depth
- max_feaatures
- min_samples_leaf
- min_sample_split
- n_estimators

In [188]:
```
def evaluate_preds(y_true, y_preds):
    """
    performs evaluation comparison on y_true labels vs. y_preds labels
    on a classification model
    """
    accuracy = accuracy_score(y_true, y_preds)
    precision = precision_score(y_true, y_preds)
    recall = recall_score(y_true, y_preds)
    f1 = f1_score(y_true, y_preds)
    metric_dict = {'accuracy': round(accuracy, 2),
                   'precision': round(precision, 2),
                   'recall': round(recall, 2),
                   'f1': round(f1, 2)}
    print(f'Acc: {accuracy * 100: .2f}%')
    print(f'Precision: {precision: .2f}')
    print(f'Recalls: {recall: .2f}')
    print(f'F1 score: {f1: .2f}')

    return metric_dict



```

In [189]:

```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

# shuffle the data
heart_disease_shuffled = heart_disease.sample(frac = 1)

# split into x and y
x = heart_disease_shuffled.drop('target', axis = 1)
y = heart_disease_shuffled['target']

# split the data into train, validation and test sets
train_split = round(0.7 * len(heart_disease_shuffled)) # 70% of data
valid_split = round(train_split + 0.15 * len(heart_disease_shuffled)) # 15% 0f data
x_train, y_train = x[:train_split], y[:train_split]
x_valid, y_valid = x[train_split:valid_split], y[train_split:valid_split]
x_test, y_test = x[valid_split:], y[:valid_split]


clf = RandomForestClassifier()
clf.fit(x_train, y_train)

# make baseline predictions
y_preds = clf.predict(x_valid)

# evaluate the classifier on validation set
baseline_metrics = evaluate_preds(y_valid, y_preds)
```

```
Acc:  82.22%
Precision:  0.81
Recalls:  0.88
F1 score:  0.85
```

In [190]:
```
np.random.seed(42)

# create a second classifier with different hyperparamters
clf_2 = RandomForestClassifier(n_estimators = 100) #n_estimators by default in this version is 100 al
clf_2.fit(x_train, y_train)

# make predictions with different hyperparameters
y_preds_2 = clf_2.predict(x_valid)

# evaluate the second classifier
clf_2_metrics = evaluate_preds(y_valid, y_preds)
```

```
Acc:  82.22%
Precision:  0.81
Recalls:  0.88
F1 score:  0.85
```

we can adjust the parameters of the model above by hand to tune it but is a lot of work

5.2 Hyperparameter tuning with RandomizedSearchCV

In [191]:

```
from sklearn.model_selection import RandomizedSearchCV

grid = {'n_estimators': [10, 100, 200, 500, 1000, 1200],
       'max_depth': [None, 5, 10, 20, 30],
       'max_features': ['auto', 'sqrt'],
       'min_samples_split': [2, 4, 6],
       'min_samples_leaf': [1, 2, 4]}

np.random.seed(42)

x =heart_disease_shuffled.drop('target', axis = 1)
y = heart_disease_shuffled['target']

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# Instantiate RandomForestClassifier

clf = RandomForestClassifier(n_jobs=1)

# setup RandomizedSearchCV
rs_clf = RandomizedSearchCV(estimator=clf, param_distributions=grid,
                            n_iter = 10, # number of models to try
                            cv =5,
                            verbose=2)

# fit the RandomizedSearchCV version of clf
rs_clf.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; tot
al time=   3.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; tot
al time=   2.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; tot
al time=   2.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; tot
al time=   2.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=1200; tot
al time=   3.0s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=100; tot
al time=   0.3s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=100; tot
al time=   0.3s
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; tot
al time=   0.4s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; tot
al time=   0.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; tot
al time=   0.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_estimators=10; total
time=   0.0s
```

```
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_estimators=10; total
time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_estimators=10; total
time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_estimators=10; total
time=   0.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_estimators=10; total
time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=10; tota
l time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=10; tota
l time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=10; tota
l time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=10; tota
l time=   0.0s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=2, min_samples_split=4, n_estimators=10; tota
l time=   0.0s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.6s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=200; tot
al time=   0.6s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=200; tot
```

```
al time=   0.5s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=4, n_estimators=200; tot
al time=   0.5s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=1000; to
tal time=   2.6s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=1000; to
tal time=   2.3s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=1000; to
tal time=   2.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=1000; to
tal time=   2.2s
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=4, n_estimators=1000; to
tal time=   2.7s
```

In [192]:

```
rs_clf.best_params_
```

Out[192]:

```
{'n_estimators': 200,
 'min_samples_split': 6,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': None}
```

In [193]:

```
# make predictions with the best hyperparameters
rs_y_preds = rs_clf.predict(x_test)

# evaluate the predictions
rs_metrics = evaluate_preds(y_test, rs_y_preds)
```

```
Acc:  81.97%
Precision:  0.77
Recalls:  0.86
F1 score:  0.81
```

5.3 Hyperparameter tuning with GridSearchCV

In [194]:
```
grid
```

Out[194]:
```
{'n_estimators': [10, 100, 200, 500, 1000, 1200],
 'max_depth': [None, 5, 10, 20, 30],
 'max_features': ['auto', 'sqrt'],
 'min_samples_split': [2, 4, 6],
 'min_samples_leaf': [1, 2, 4]}
```

In [195]:
```
# let's reduce the search space for gridserchcv/comparing with the best param

grid_2 = {'n_estimators': [100, 200, 500],
          'max_depth': [None],
          'max_features': ['auto', 'sqrt'],
          'min_samples_split': [6],
          'min_samples_leaf': [1, 2]}
```

In [196]:
```
from sklearn.model_selection import GridSearchCV, train_test_split

np.random.seed(42)

x =heart_disease_shuffled.drop('target', axis = 1)
y = heart_disease_shuffled['target']

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# Instantiate RandomForestClassifier

clf = RandomForestClassifier(n_jobs=1)

# setup GridSearchCV
gs_clf = GridSearchCV(estimator=clf,
                      param_grid=grid_2,
                             cv =5,
                             verbose=2)

# fit the GridSearchCV version of clf
gs_clf.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.4s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.7s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.6s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.7s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.3s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.6s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.5s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.5s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   2.2s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.6s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.5s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.4s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.6s
```

```
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.7s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.8s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.7s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.9s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.4s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.3s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.5s
[CV] END max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.5s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.6s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
```

```
otal time=   1.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=6, n_estimators=500; t
otal time=   1.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=100; t
otal time=   0.1s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=200; t
otal time=   0.4s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.2s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.3s
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_estimators=500; t
otal time=   1.6s
```

In [197]:
```
gs_clf.best_params_
```

Out[197]:
```
{'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 200}
```

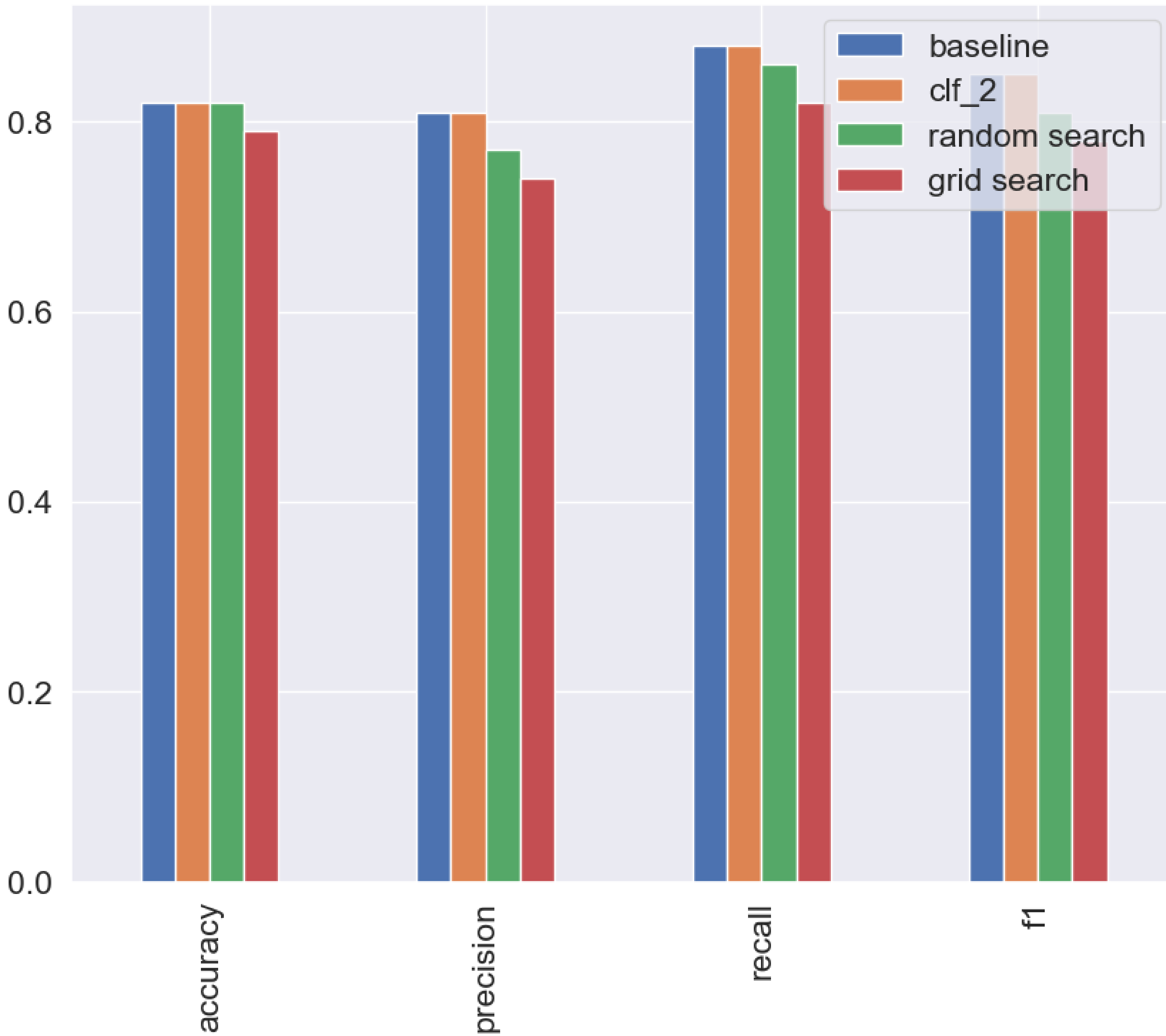
In [198]:
```
gs_y_preds = gs_clf.predict(x_test)

# evaluate the predictions
gs_metrics = evaluate_preds(y_test, gs_y_preds)
```

```
Acc:  78.69%
Precision:  0.74
Recalls:  0.82
F1 score:  0.78
```

let's compare our different models metrics

In [199]:

```
compare_metrics = pd.DataFrame({'baseline': baseline_metrics,
                                'clf_2': clf_2_metrics,
                                'random search': rs_metrics,
                                'grid search': gs_metrics})
compare_metrics.plot.bar(figsize = (10, 8));
```



baseline
clf_2
random search
grid search
0.0
0.2
0.4
0.6
0.8
accuracy
precision
recall
f1

6. Saving and loading trained machine learning models.

Two ways to save and load learning models:

1. with python's 'pickle' module
2. with the 'joblib' module

Pickle

In [200]:
```
import pickle

# save an existing model to fileS
pickle.dump(gs_clf, open('gs_random_forest_model_1.pkl', 'wb'))
```

In [201]:
```
# load a saved model

loaded_pickle_model = pickle.load(open('gs_random_forest_model_1.pkl', 'rb'))
```

In [202]:
```
# make some predictions
pickle_y_preds = loaded_pickle_model.predict(x_test)
evaluate_preds(y_test, pickle_y_preds)
```

```
Acc:  78.69%
Precision:  0.74
Recalls:  0.82
F1 score:  0.78
```

Out[202]:
```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

Joblib

In [203]:
```
from joblib import dump, load

# save model to file
dump(gs_clf, filename = 'gs_random_random_model_1.joblib')
```

Out[203]:
```
['gs_random_random_model_1.joblib']
```

In [204]:
```
# import a saved joblib model
loaded_joblib_model = load(filename = 'gs_random_random_model_1.joblib')
```

In [205]:
```
# make and evaluate joblib predictions
joblib_y_preds = loaded_joblib_model.predict(x_test)
evaluate_preds(y_test, joblib_y_preds)
```

```
Acc:  78.69%
Precision:  0.74
Recalls:  0.82
F1 score:  0.78
```

Out[205]:
```
{'accuracy': 0.79, 'precision': 0.74, 'recall': 0.82, 'f1': 0.78}
```

7. Putting it all togeteher

In [206]:
```
data = pd.read_csv('car-sales-extended-missing-data.csv')
data
```

Out[206]:

	Make	Colour	Odometer (KM)	Doors	Price
0	Honda	White	35431.0	4.0	15323.0
1	BMW	Blue	192714.0	5.0	19943.0
2	Honda	White	84714.0	4.0	28343.0
3	Toyota	White	154365.0	4.0	13434.0
4	Nissan	Blue	181577.0	3.0	14043.0
...	...	...	...	...	...
995	Toyota	Black	35820.0	4.0	32042.0
996	NaN	White	155144.0	3.0	5716.0
997	Nissan	Blue	66604.0	4.0	31570.0
998	Honda	White	215883.0	4.0	4001.0
999	Toyota	Blue	248360.0	4.0	12732.0

1000 rows × 5 columns

In [207]:
```
data.dtypes
```

Out[207]:
```
Make             object
Colour           object
Odometer (KM)   float64
Doors           float64
Price           float64
dtype: object
```

In [208]:
```
data.isna().sum()
```

Out[208]:
```
Make            49
Colour          50
Odometer (KM)   50
Doors           50
Price           50
dtype: int64
```

Steps we want to do (all in one cell):

1. fill missing data
2. convert the data to numbers
3. build a model on the data

In [1]:
```
# getting started
import pandas as pd
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# modelling
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV

# set random seed
import numpy as np
np.random.seed(42)

# import data and drop rows with missing labels
data = pd.read_csv('car-sales-extended-missing-data.csv')
data.dropna(subset=['Price'], inplace=True)

# define different features and transform pipeline
categorical_features = ['Make', 'Colour']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

door_feature = ['Doors']
door_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=4))
])

numeric_features = ['Odometer (KM)']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])

# setup preprocessing steps (fill missing values, then convert to numbers)
preprocessor = ColumnTransformer(
                        transformers = [
                            ('cat', categorical_transformer, categorical_features),
                            ('door', door_transformer, door_feature),
                            ('num', numeric_transformer, numeric_features)
                        ])

```

```python
# creating a preprocessing and modeling pipeline
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('model', RandomForestRegressor())])

# split data
x = data.drop('Price', axis = 1)
y = data['Price']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# fit and score the model
model.fit(x_train, y_train)
model.score(x_test, y_test)

```

Out[1]: 0.22188417408787875

it's also possible to use GridSearchCV or RandomizedSearchCV with our Pipeline

In [4]:

```
# use GridSearchCV with our regression Pipeline
from sklearn.model_selection import GridSearchCV

pipe_grid = {
    'preprocessor__num__imputer__strategy': ['mean', 'median'],
    'model__n_estimators': [100, 1000],
    'model__max_depth': [None, 5],
    'model__max_features': ['auto'],
    'model__min_samples_split': [2, 4]
}

gs_model = GridSearchCV(model, pipe_grid, cv=5, verbose=2)
gs_model.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=mean; total time=   0.4s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=mean; total time=   0.3s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=mean; total time=   0.4s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=median; total time=   0.4s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=median; total time=   0.4s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=median; total time=   0.5s
[CV] END model__max_depth=None, model__max_features=auto, model__min_samples_split=2, model__n_estimat
ors=100, preprocessor__num__imputer__strategy=median; total time=   0.4s
```

In [5]:

```
gs_model.score(x_test, y_test)
```

Out[5]:

```
0.3339554263158365
```

In [ ]:

```

```